http://dx.doi.org/10.14400/JDC.2015.13.6.165

SETL을 이용한 소프트웨어의 컴포넌트 기반 융복합 개발 방법

유홍준*, 양해술**
호서대학교 벤처대학원 정보경영학과 박사과정* 호서대학교 벤처대학원 정보경영학과 교수**

Component-based Convergence Development Method of Software using SETL

Hong-Jun Yoo*, Hae-Sool Yang**
Doctor Candidate, Dept. of Information Management, Graduate School of Venture, Hoseo University*
Professor, Dept. of Information Management, Graduate School of Venture, Hoseo University**

요 약 정보 시스템을 구현하는 프로그램을 설계하는 방법은 Flowchart에서 UML의 Activity Diagram에 이르기까지 다양하다. 하지만, 이제까지 개발된 프로그램 설계 도구와 방법은 프로그램 코딩 도구와 방법에 비해서 상대적으로 효율적이지 않았다. 또한 프로그램 설계와 코드 간의 쌍방향 절환이 용이하지 않아 개발 생산성과 유지보수성을 개선하는 데 한계가 있었다. 따라서 본 연구에서는 컴포넌트 기반의 SOC(Structured Object Component)을 지원하는 설계 및 코딩 융복합 자동화 도구 SETL(Structured Efficiency TooL)의 개발을 통해 프로그램 설계와 코딩 단계를 융복합하여 병렬적인 작업이 가능하도록 하는 융복합 개발 방법을 제안한다. 즉, SETL을 사용하면 프로그램 설계와 코딩 간의 절환을 거의 실시간으로 수행할 수 있어, 소프트웨어 개발 공정 단계간의 격차를 해소하여 개발 생산성 및 유지보수성을 극대화 할 수 있다.

주제어 : SETL, SOC, UML, 소프트웨어 재공학, 융복합 개발

Abstract Methods to design programs which implement IT systems have been developed in various forms from flowchart to activity diagram of UML. However, program design tools and methods developed so far have not been efficient comparing to program coding tools and methods. In addition, Program design methods and tools developed until now have been difficult to support the bidirectional conversion between program design and coding, and the improvement of development productivity and maintainability. Therefore, in this study, we propose Convergence Development Method to enable working with wide bandwidth through fusing the program design and coding phase by using SOC and supporting tool named SETL which automatizes the convergence of design and coding. Thus, by using SETL, it is expected that the efficiency gap between the program design and coding phase is reduced, and development productivity and maintainability is increased.
key words.

Key Words : SETL, SOC, UML, Software Reengineering, Convergence Development

Received 2 April 2015, Revised 6 May 2015
Accepted 20 June 2015
Corresponding Author: Hae-Sool Yang
(Graduate School of Venture, Hoseo University)
Email: hsyang@hoseo.edu

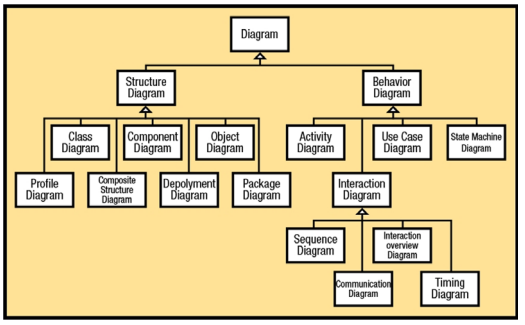
ISSN: 1738-1916

1. 서론

1921 년 Frank Gilbreth이 "flow process chart"방법을 처음 소개하고 [1] 1947 년에 Herman Goldstine과 John von Neumann이 미발표 보고서인 'Planning and coding of problems for an electronic computing instrument, Part II, Volume 1 " [2]에서 컴퓨터 프로그램 설계용 플로우차트를 처음 소개한 이래 1960 년대 중반 이후에 구조화 프로그래밍 개념이 탄생하고 [3, 4] 1970 년대 들어 급속히 성장하였다[5].

이를 계기로 1970 년대와 1980 년대에 걸쳐 Hamilton - Zeldin diagram [6], NS diagram [7], TS charts [8], Witty chart [9], Ferstl diagrams [10], PF [11], LCP flowchart [11], CP hierarchical [11], HCP [13], SPD [13], YAC [13], PAD [12], Structured D - chart [14], DSD, PSD, Warnier - Orr Diagram [15, 16], Action Diagram [17] 등의 프로그램 설계 모델링과 관련한 다양한 관점의 접근 방법이 출현했다.

1990 년대에는 객체지향 방법 기반의 UML (Unified Modeling Language)이 탄생하고 프로그램의 흐름을 보여주는 Activity Diagram을 비롯한 9 개의 모델링 방법이 등장하여 몇 차례의 개정을 거쳐 [Fig. 1]과 같이 7 개의 Structure Diagram과 7 개의 Behavior Diagram 등 총 14 종류의 모델링 방법으로 확장되었다[18, 19, 20].



[Fig. 1] UML diagrams for integrated modeling

2000 년대 들어 모바일과 융복합 기술의 급속한 발전과 함께, 애자일 프로그래밍 기술 등 소프트웨어 개발 환경도 근본적으로 변화하고 있다. 하지만 프로그램 설계 방법은 더 이상 혁신이 일어나지 않고 있으며, 오늘날 세계적으로 De Facto 표준으로 가장 큰 영향력을 미치고 있는 UML의 Activity Diagram은 프로그램의 부분적인 알고리즘의 표현에 한계를 보이고 있는 실정이다.

실제로 우리나라에서 2000 년부터 2014 년까지 15년간에 걸쳐 진행된 321 건의 공공 정보화 프로젝트를 분석한 결과, UML의 사용 비율이 높았지만 프로그램의 설계 시방서에서 프로그램 알고리즘 전체를 설계한 사례는 한 건도 없었다.

이처럼 소프트웨어 공학 연구는 지난 40 년 이상의 기간 동안 진화해 왔지만, 소프트웨어 위기는 아직도 현재 진행형인 상태이다[21].

따라서 급속히 복잡화 되는 소프트웨어 개발 환경을 효율적으로 지원하기 위해 재공학 기술을 기반으로 하는 SETL의 개발을 통해 소프트웨어 융복합 개발 방법을 제시하고자 한다.

2. 관련 연구

2.1 프로그램 설계 관련 기존 연구 현황

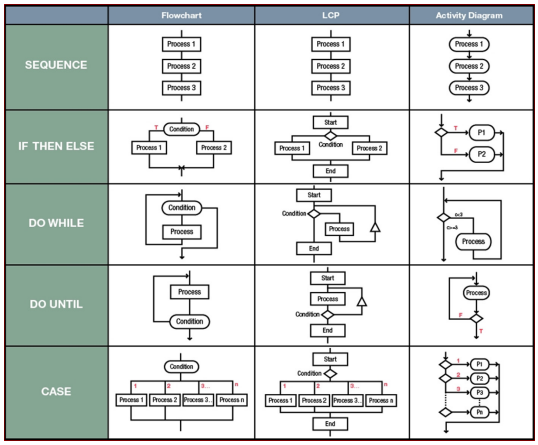
알고리즘을 기반으로 하는 기존의 프로그램 설계 모델링 방법 중 상호 비교가 용이한 주요 방법을 추출하여 분류하면 크게 흐름 중심 방법과 구조 중심 방법으로 구분이 가능하다.

흐름 중심 방법으로는, Flowchart, LCP (Logical Conception of Program), Activity Diagram 등의 3 가지를 들 수 있다.

구조 중심 방법으로는, NS (Nassi-Shneiderman) chart, DSD (Design Structure Diagram), Action Diagram, PAD (Problem Analysis Diagrams), HCP (Hierarchical and Compact description chart) , SPD (Structured Programming Diagram), YAC (Yet Another Control Chart) 등의 7 가지가 대표적이다.

특히, 흐름 중심의 설계 모델링 방법은 2000 년대 들어 세계적으로 UML의 Activity Diagram이 사용 비율 면에서 압도적인 영향을 미치고 있다.

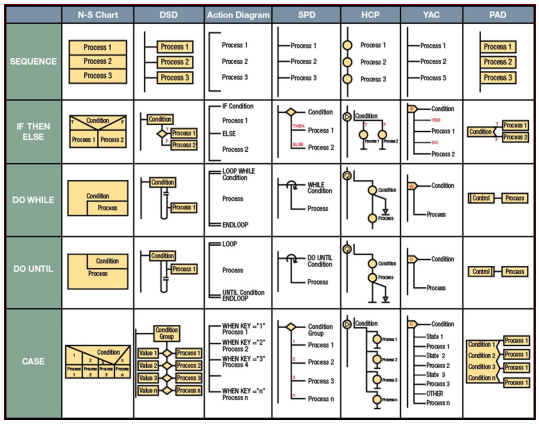
흐름 중심의 주요 프로그램 설계 모델링 방법을 서로 비교하면 [Fig. 2]와 같다.



[Fig. 2] Comparison among Diagramming Methodologies of Flow-oriented Charts

구조 중심 방법으로는 구조적 분석과 구조적 설계 방법이 주류를 이루던 1980 년대와 1990 년대 초반까지 다양한 기법이 출현하였다.

구조 중심 방법의 주요 프로그램 설계 모델링 방법을 서로 비교하면 [Fig. 3]과 같다.



[Fig. 3] Comparison among Diagramming Methodologies of Structure-oriented Charts

그러나 구조 중심 방법은 1990년대 중반 이후에 개발 환경이 구조적 방법에서 객체지향 컴포넌트 방법으로 진화함에 따라 급속히 하향곡선을 그리기 시작했다. 이후 2015년 현 시점까지 일본 등 세계 일부 지역을 제외하고는 흐름 중심인 UML의 Activity Diagram이 주도권을 확보하고 있는 실정이다.

2.2 기존의 프로그램 개발 방법의 문제점

현재 가장 큰 영향력을 가지고 있는 UML의 Activity Diagram을 살펴보면, Fork, Join, Transition, Branch 등의 표현을 사용하여 프로그램을 설계한 후 프로그램 소스로 코딩하는 전형적인 흐름 중심의 방법이다.

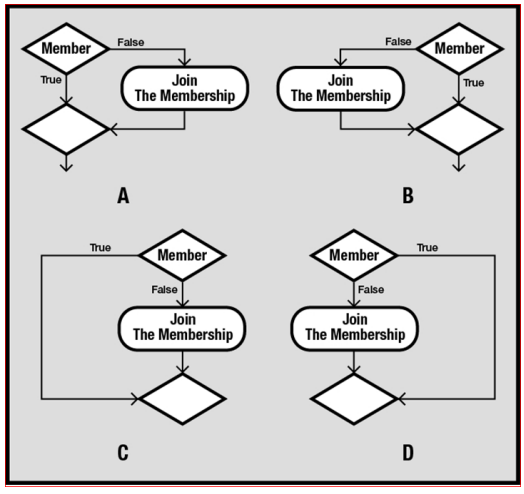
하지만, 알고리즘의 표현에 있어서 많은 중요한 결함을 가지고 있다.

첫째, 코딩 작업보다 Activity Diagram의 작업에 소요되는 시간이 훨씬 길다. 이는 설계 작업이 코딩 작업보다 시간 효율성 측면에서 불리함을 의미한다.

둘째, 프로그램 Activity를 폐쇄된 형태의 도형으로 표현하여 알고리즘을 프로그램 코드와 1:1 대응시킬 수 있을 정도로 상세히 표현하는 것이 불가능하며, 개요 파악도 어렵다.

셋째, Activity Diagram의 기술 시 공간을 차지하는 밀도가 프로그램 코드보다 훨씬 낮아 표현 공간의 낭비가 많다. 이는 한 눈에 파악할 수 있는 정보에 한계를 초래하여 작성한 설계 내역의 이해도를 저하시키는 요인으로 작용한다.

넷째, 어떤 Activity Diagram도 [Fig. 4]를 통해 알 수 있는 바와 같이 정형화 된 흐름 패턴을 가지고 있지 않다. 이는 거의 무한대의 패턴 다양화를 초래하여 컴퓨팅 사고(Computational Thinking)의 기반이 되는 패턴 인식을 불가능하게 한다.



[Fig. 4] Different patterns of a same algorithm by Activity Diagram

다섯째, 설계와 코딩 공정이 분리되어, 언제나 설계가 끝난 후에 코딩 작업을 진행하는 형태의 직렬형 작업 방법을 사용해야 한다. 이는 설계와 코딩을 병행하여 진행하는 형태의 병렬형 작업 방법을 채택할 수 없게 하는 제약요건으로 작용한다.

이러한 근본적인 문제로 인해 기존의 프로그램 설계 방법은 급속히 발전하는 소프트웨어 관련 기술을 지원하지 못하고 있는 상황이다. 결과적으로 기존의 프로그램 설계 방법은 규모에 관계없이 각종 정보화 프로젝트에서 알고리즘을 중심으로 하는 프로그램 설계를 실종시키는 결정적인 요인으로 작용해왔다.

2.3 기존 방식의 한계 극복 방안

Activity Diagram을 포함한 기존 기술의 문제점을 해결하기 위해서는 아래의 다섯 가지 사항에 대한 고려가 필요하다.

첫째, 요구사항 분석을 바탕으로 이해하기 쉽게 도해하여 코딩을 지원하는 설계 수준의 작업이 코딩 수준의 작업보다 시간효율성이 뛰어나야 한다.

둘째, 프로그램 Activity를 개방된 형태로 표현하여 설계 내역과 코딩 내역이 1:1로 완벽하게 대응할 수 있도록 하고, 추상화 사다리 구성을 통해 개요와 상세를 동시에 파악할 수 있도록 해야 한다.

셋째, 프로그램 설계 내역의 기술 밀도가 코드의 기술 밀도와 크게 차이나지 않아야 한다.

넷째, 특정한 알고리즘에 대해서는 어떠한 경우에도 패턴이 일정하게 나타나야 한다.

다섯째, 설계와 코딩 작업을 병행적으로 수행하여 공정 단계간의 간극을 제거할 수 있어야 한다.

이러한 조건을 충족시킬 수 있도록 먼저 CPD(Control Pattern Diagram)이라는 이름으로 SOC(Structured Object Component)을 고안해내었으며, 이를 지원하는 자동화 도구인 SETL을 개발하기 시작했다 [22]. <Table 1>은 SOC과 기존의 주요 프로그램 설계 모델링 방법과의 차이를 나타낸 것이다.

본 연구에서는, SOC을 지원하기 위해 장기간에 걸쳐서 개발해 온 SETL(Structured Efficiency TooL)의 원리와 이를 사용하여 설계와 코딩을 병행적으로 수행하는 융복합 개발 방법을 제안하고자 한다.

3. SETL의 기본 원리

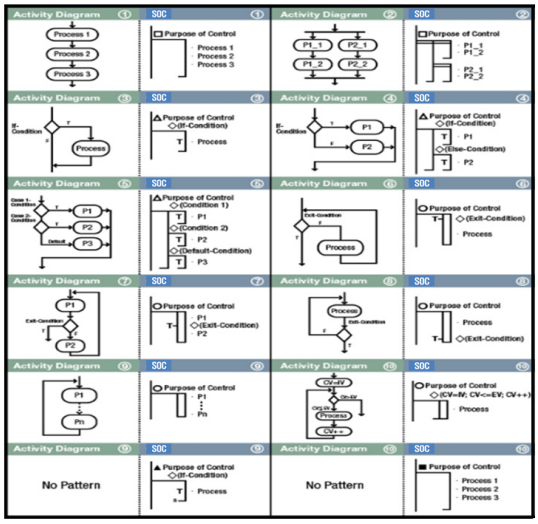
3.1 SETL이 지원하는 SOC의 개요

SOC을 지원하는 SETL을 이용하면 컴포넌트 기반의 조립식 설계를 실현할 수 있으며, 프로그램 설계와 코딩을 병행적으로 수행할 수 있어 소프트웨어 개발 및 유지보수 효율성을 획기적으로 향상시킬 수 있다.

〈Table 1〉 Comparing SOC and other diagrams

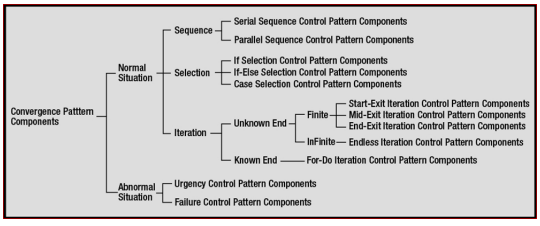
Comparing items	Flowchart	Activity Diagram	NS chart	PAD	HCP	SPD	YAC	SOC
Modeling approach	Flow-Oriented	Flow-Oriented	Structure-Oriented	Structure-Oriented	Structure-Oriented	Structure-Oriented	Structure-Oriented	Structure-Oriented
Describing Method	Closed	Closed	Closed	Closed	Open	Open	Open	Open
Identifying Control Structure	Difficult	Difficult	Easy	Easy	Easy	Easy	Easy	Easy
Detailed Expression	Difficult	Difficult	Plain	Difficult	Easy	Easy	Easy	Easy
Describing Density	Low	Low	High	Plain	High	High	High	High
Language Support	Difficult	Plain	Difficult	Difficult	Difficult	Difficult	Difficult	Easy
Goto Support	Yes	Yes	No	No	Yes	No	No	Yes
Workflow Modeling	No	Yes	No	No	No	No	No	No
Purpose Expression	No	No	No	No	Yes	No	No	Yes
Pattern Recognition	No	No	No	No	No	No	No	Yes
Hierarchy Expression	Difficult	Difficult	Difficult	Plain	Plain	Plain	Plain	Easy
Algorithm Tracing	Plain	Plain	Easy	Easy	Plain	Plain	Plain	Easy
Analysis Support	No	Yes	No	No	No	No	No	No
Documentation	Difficult	Plain	Plain	Plain	Easy	Easy	Easy	Easy
Parallel Processing Support	No	Yes	No	No	No	No	No	Yes
Recursion Support	No	No	Yes	No	No	No	No	Yes
Assembly Automation	No	No	No	No	No	No	No	Yes

우선, SOC의 모델링 방법을 UML의 Activity Diagram과 비교하면 [Fig. 5]와 같다.



[Fig. 5] SOC compared to UML Activity Diagram

SETL은 SOC을 지원하기 위해 문제를 해결하기 위한 알고리즘 패턴 부품을 [Fig. 6]과 같이 크게 정상적인 상황(Normal Situation)과 비정상적인 상황(Abnormal Situation)으로 구분하여 대응한다.



[Fig. 6] Types of SETL Program Components

SETL의 정상 상황 처리 기능은 계획대로 실행할 수 있는 상황의 알고리즘 처리를 위한 기능이며, 비정상 상황 처리 기능은 환경의 변화에 따라 시스템에 부정적인 영향을 줄 수 있는 비정상적인 상황의 알고리즘 처리를 위한 기능을 의미한다.

또한 비정상 상황 처리 기능은 시스템 자체적으로 신속하게 대응 가능한 비상 상황(Urgency Situation) 처리 기능과 외부의 도움을 받는 것을 포함한 보다 체계적인 대응을 위한 이상 상황(Failure Situation) 처리 기능으로 구분할 수 있다.

이를 크게 정상계, 비상계 및 이상계로 분류하여 각각의 처리 목적에 따라 기능과 SETL이 취급하는 설계 컴포넌트 별로 종류를 세분화 한다.

3.2 SETL의 동작 원리

소프트웨어의 개발을 문제 해결의 관점에서 접근하면 "분석 → 설계 → 구현"의 과정은 "문제 인식 → 해결책 강구 → 문제 해결"의 과정으로 바꿔서 생각할 수 있다.

문제 인식이 이미 이루어져 있다는 것을 전제하면, 개발 생산성과 유지보수성을 극대화해주기 위해서는 해결책 강구와 문제 해결의 단계간의 간극의 극소화가 중요하다.

또한 우리가 문제를 해결할 때, 언제나 계획을 세워서 해결책을 강구해내고 나서 문제 해결을 하기만 하는 것이 아니다. 순간적으로 번득이는 아이디어가 떠올라 난해할 것 같았던 문제를 현장에서 쉽게 해결하는 경우도 많다. 따라서 개발 및 유지 보수 과정에서의 효율성 제고를 위해서는 설계 단계와 구현 단계간의 유연한 순환에 의한 진화적인 성숙이 바람직하다.

더 나아가서는 프로젝트에 참여하는 여러 구성원들이 설계와 코딩을 각각 나눠서 수행하더라도 그것들을 실시간으로 융복합하여 병행적으로 작업할 수 있도록 병렬적인 개발 방법을 적용할 필요가 있다.

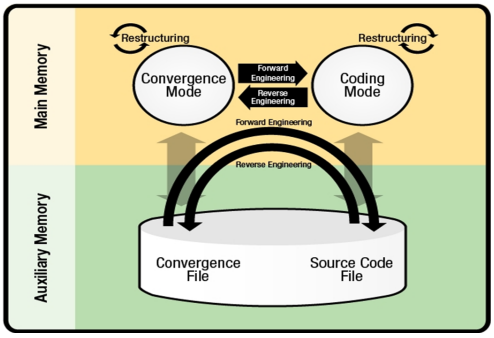
SETL을 개발하던 초기에는 개념적으로 순공학, 역공학 및 재구조화를 모두 지원하고자 목표하였으나, 실제로는 순공학 기능과 부분적인 재구조화 기능에 한정되었다. 또한 지원 언어도 Visual BASIC에 국한되었다.

하지만, 컴퓨터 기술의 발전에 따라 언어가 C언어 계열과 Java언어 계열로 대세가 결정됨에 따라, Visual BASIC과 같은 일정한 형식을 가진 언어에서 발전하여 자유 양식으로 작성이 가능한 C 및 Java 언어 계열에 대한 지원이 가능하도록 개발 작업을 진행했다.

그 결과, [Fig. 7]에 나타낸 바와 같이, 코딩 모드와는 별도로 융복합 모드를 가지고 있으며, C언어 및 Java언어도 지원하는 혁신적인 SETL을 개발하는데 성공하였다.

코딩 모드(Coding Mode)는 전통적인 소스 코드 에디터와 동일한 방식으로 작용하지만, 융복합 모드(Convergence Mode)는 개요설계와 상세설계를 융복합

한 형태로 개요와 상세간의 추상화 사다리(Ladder of Abstraction)를 구성한다.



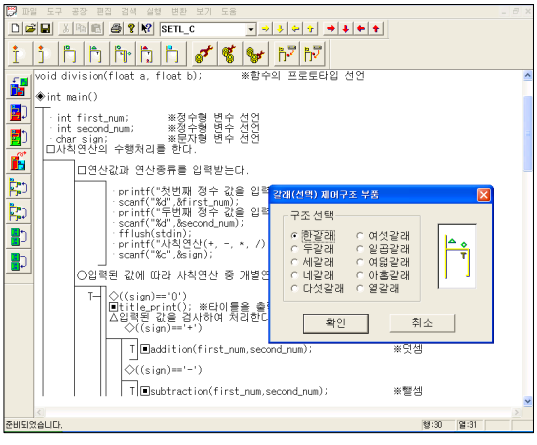
[Fig. 7] Processing Modes of SETL

컴포넌트 기반의 새롭게 혁신된 SETL의 동작은 이전과는 달리 기본적으로 순공학(Forward Engineering), 역공학(Reverse Engineering) 및 재구조화(Restructuring) 기술을 모두 지원하는 특징을 가진다.

SETL의 동작 원리를 구체적으로 나타내면 다음과 같다.

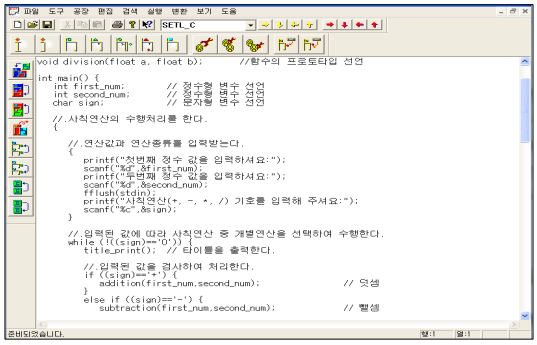
첫째, SETL을 실행하면 기본적으로 일반 소스 코드 에디터처럼 편집하는 기능이 작동한다(소스 편집 기능).

둘째, 설계를 융복합적으로 수행하려면 [Fig. 8]과 같이 해당 에디터 모드에서 바로 컴포넌트 식 SOC의 조립•분해를 툴바(Toolbar)를 이용하여 단 한 번의 클릭만으로 행할 수 있다(조립과 분해 기능).



[Fig. 8] Example of Assembly and Disassembly Work in Convergence Mode of SETL

셋째, 융복합 모드에서의 설계 작업을 완료하면 모드 변환 기능을 사용하여 [Fig. 9]와 같이 거의 실시간으로 완벽한 소스 코드를 생성해낼 수 있다(순공학 기능).



[Fig. 9] Example of Forward Engineering Work of SETL

넷째, 설계 작업을 하는 과정에서도 코딩 모드로 절환하여 자연스럽게 코딩을 할 수 있으며, 소스 코드는 즉각 융복합 모드로의 절환을 통해 [Fig. 8]과 같은 상태로 복귀하여 개요 및 상세 설계 모델을 재생해 낼 수 있다(역공학 기능).

다섯째, 설계 모델로 재생한 내역은 추상화, 구체화, 추상화 범위 확장 및 범위 축소 등의 기능을 적용하여 재구조화 할 수 있다(재구조화 기능).

여섯째, SETL은 비체계적으로 작성한 소스 코드를 읽어서 설계 모델을 재생할 때, 비구조적이거나 품질이 저하된 부분을 자동으로 인식하여 개선해준다(자동 개선 기능)

상기와 같은 기능을 중심으로 하는 동작을 통해 SETL은 소프트웨어 개발자가 설계와 코딩을 병행하여 융복합적으로 수행하는 것을 지원해줌으로써 융복합 개발(Convergence Development)을 실현한다.

3.3 SETL의 특징

프로그램 설계와 코드 간의 상호 변환 기능은 SETL만의 전유물은 아니며, PAD (Problem Analysis Diagram)를 지원하는 소프트웨어인 WebMakePAD 등과 같은 도구도 이미 적용하고 있다[23].

하지만, PAD를 지원하는 WebMakePAD의 경우, 구조화 된 C언어에서 적용 시 설계와 코드 간의 절환이 제한적인 조건하에서만 가능하고, Java와 같은 객체지향

언어에는 아직 적용하지 못하고 있는 상태이다 [24].

아울러 자유로운 코드 개선이나 재구조화가 불가능한 상태이다.

이에 비해 본 연구에서 제안하는 SETL은 C를 중심으로 하는 구조적 프로그래밍 언어는 물론 Java를 중심으로 하는 객체지향 언어에 모두 적용이 가능하다. 또한, 설계와 코드 간의 순간 절환은 물론, 복잡하게 작성한 소스 코드 구조의 개선이나 컴포넌트 기반의 조립과 분해를 통한 재구조화를 용이하게 지원해주는 등 작업의 효율성을 극대화 한 것이 특징이다.

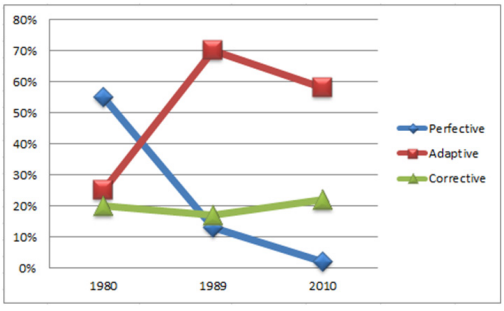
4. SETL을 이용한 융복합 개발의 방법

4.1 융복합 개발을 위한 전제 사항

최근 급속한 프로그램 개발 환경의 변화에 따라 새로 개발한 소프트웨어와는 별도로 기존에 구축한 소프트웨어 시스템의 고도화, 개선 등을 중심으로 하는 유지보수성 프로젝트가 급증하고 있다.

이러한 유지보수성 프로젝트는 현행 시스템의 분석을 위해 기존에 이미 구축한 시스템의 소스 코드에 대한 이해가 필수적이다. 하지만, 관련 연구에 따르면 유지보수 담당자가 프로그램을 이해하는데 걸리는 시간이 7.5시간일 때, 실제 프로그램 변경에 소요하는 시간은 0.5시간에 불과하다는 결과를 제시하고 있다 [25].

또한 시대별로 Lientz / Swanson, DiNardo [26]와 Jean- Marc Desharnais / Alain April [27] 등이 조사한 결과에 따르면, [Fig. 10]과 같이 환경 변화에 따른 Adaptive Maintenance의 비율이 1980 년대 말 이후 현재에 이르기까지 가장 높은 것으로 나타났다.



[Fig. 10] Trends of Adaptive Maintenance Rates in Software

구축한 소프트웨어 시스템 자체에 결함이 없어도 환경의 변화에 따라 시스템의 수정과 보완을 해야 하는 비율이 높아지고 있다. 그로 인해 기존에 구축한 소프트웨어 시스템의 소스 코드에 대한 이해가 중요하다.

하지만 대부분의 프로젝트에서는 분석을 완료한 후, 알고리즘 레벨의 상세 설계 없이 바로 코딩에 들어가고 있다. 따라서 일단 시스템 구축을 완료하면 유지보수 시에 시스템을 설계 단계에서 이해하지 못하고 코드 자체만을 가지고 이해해야 하는 문제가 발생한다.

본고에서 제시하는 융복합 개발이라는 의미는 단순히 개발을 병행적으로 수행한다는 개념을 넘어서 유지보수와 개발도 병행적으로 수행할 수 있다는 것을 전제로 한다.

개발 측면에서는 기본적으로 순공학 기술을 기준으로 하지만, 유지보수 측면에서는 기본적으로 역공학 기술을 기준으로 하는 것을 전제로 한다.

4.2 SETL을 이용한 융복합 개발 방법의 기대 효과

프로그램을 이해하는 데 걸리는 시간을 단축하기 위해서는, 소스 코드 수준이 아니라 설계 단계의 수준에서 이해할 수 있도록 대응할 필요가 있다. 프로그램의 설계 사양을 추상화 수준이 높은 개요 설계 부분으로부터 알고리즘의 상세한 표현이 가능한 수준까지 작성하는 것이 중요하다.

SETL을 사용하면 융복합 모드에서 설계와 구현 내역을 융합적으로 파악할 수 있기 때문에 프로그램의 소스를 이해하는 데 걸리는 시간을 크게 줄일 수 있다.

소스 코드는 해안가에 자갈이 흩어져 있는 것과 같다. 따라서 분산된 형태로 구성한 상태에서는 구조의 이해가 쉽지 않고 제어 흐름의 해석도 용이하지 않은 성질을 가지고 있다.

하지만, SETL의 융복합 모드에서는 제어 구조를 파이프라인의 형태로 표현하므로 제어 흐름에 대한 이해가 매우 용이하다. 즉 융복합 모드에서는 전자 회로를 추적하는 것처럼 알고리즘을 쉽게 추적하여 이해할 수 있다.

이처럼 SETL을 이용하면 설계단계에서의 작업이 용이하다. 또한, 코딩 단계에서 임의로 무질서하게 편집한 내용이라 하더라도, SETL이 프로그램의 제어구조를 인식하여 자동적으로 재구조화 하여 역공학으로 설계 절환

을 할 수 있다.

이로 인해, SETL을 사용하여 개발 및 유지보수 작업을 수행하면 설계자와 개발자가 다르더라도 거의 실시간으로 작업을 통합할 수 있어 병렬적인 개발 작업이 용이해진다.

특히, 복잡한 형태의 알고리즘을 간소화하고 알고리즘에 대한 이해도를 향상시켜 줌으로써 개발 및 유지보수 공정 작업의 효율성을 급속히 향상시켜 주는 성과를 창출한다.

또한 알고리즘의 수정 및 보완을 통해 오류의 확률을 급속히 낮출 수 있어 알고리즘의 정확도를 높여주는 효과를 기대할 수 있다.

5. 융복합 개발 방법의 품질 측정 결과

5.1 융복합 개발 방법의 품질 측정 기준

프로그램의 소스는 유한한 단계를 거쳐 문제를 해결하기위한 절차를 구성하는 알고리즘을 형성한다. 세심하게 정의한 명백한 규칙들의 집합을 알고리즘이라고 할 때 유한한 단계를 거쳐 문제를 해결하기위한 과정이다.

아울러 알고리즘은 문제를 해결하기 위해 여러 단계의 유한 집합을 구성한다.

알고리즘의 품질을 측정하기 위해서는 다음과 같은 측정 기준의 적용을 필요로 한다.

- 정확도 : 가장 기본적인 갖추어야 할 기준이다.
- 작업량과 처리속도 : 부하가 적고 처리 속도가 빠를수록 좋은 알고리즘이다.
- 기억 장소의 사용 : 가능한 한 적은 양의 기억 장소를 사용하는 것이 좋은 알고리즘이다.
- 단순성 : 단순할수록 생성 및 수정을 위한 알고리즘을 쉽게 이해할 수 있다.
- 최적성 : 같은 조건이면 연산의 수가 적을수록 좋은 알고리즘이다.

위의 측정 기준에서 처리량과 처리속도와 최적성은 시간 효율성에 속하며 기억 장소의 사용량은 공간 효율성에 속한다. 또한 단순성과 정확성은 코드 효율성에 속한다.

프로그램은 유한개의 알고리즘으로 구성되기 때문에 유지보수의 효율성을 높이기 위해 기존에 구축한 소프트웨어 시스템을 형성하기위한 프로그램의 알고리즘에 대한 이해를 어떻게 효율적으로 할 것인지가 중요한 열쇠의 역할을 한다. 즉, 일반적으로는 코드 효율성의 관점에서의 접근이 중요하다.

하지만, SETL은 설계 단계에서 용이하게 알고리즘을 파악할 수 있기 때문에, 코드 효율성 보다는 시간 효율성 측면에서의 측정 기준 마련이 필요하다.

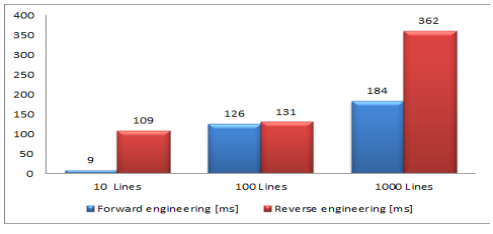
시간효율성 관점에서의 측정기준은 밀리세컨드(ms) 단위를 초로 환산하고, Visual C++언어에서 순공학 또는 역공학 변환작업 시작 직전의 시간과 변환작업 완료 직후의 시간을 측정한 후 시작 시간과 종료 시간의 클럭 차이를 밀리세컨드 단위로 계산해 내는 방법으로 마련할 수 있다.

본 연구에서는 계산의 객관성을 제고하기 위해 "time.h" 헤더 파일을 include하고 clock()함수를 프로그램에 삽입하여 측정하도록 기준을 마련하여 적용하였다.

5.2 SETL을 이용한 융복합 개발 품질 측정 결과

시간 효율성 측면의 품질 측정 기준을 마련함에 따라 순공학과 역공학 기술을 적용할 때 SOC 설계내역을 C언어 및 C++ 언어 소스 코드로 설계 단계와 코딩 단계를 병렬적으로 수행한 후 SOC 설계로부터 소스 코드로 순공학 변환하거나, 소스 코드를 SOC 설계 내역으로 역공학 변환할 때의 시간 효율성을 측정하여 검증하였다.

SETL을 사용하여 C언어와 C++언어 프로그램을 융복합 모드와 코딩 모드에서 각각 별도로 작업한 내용을 10라인 수준의 프로그램부터 1000라인 수준의 프로그램에 이르기까지 10^1에서부터 10^3 라인까지의 크기의 파일을 각각 순공학 및 역공학 방법으로 변환한 결과, 평균 [Fig. 11]과 같은 측정 결과를 얻었다.



[Fig. 11] Time Efficiency of Forward and Reverse Engineering Work of SETL

측정의 객관성을 확보하기 위해 변환 프로그램 내부에 시간을 측정하는 기능을 삽입하여 객관적이고 정량적인 측정을 수행하였다.

측정 결과를 통해서 알 수 있는 바와 같이, SETL은 융복합 모드와 코딩 모드를 자유자재로 절환하면서 효율적으로 개발 및 유지보수 작업을 수행할 수 있도록 지원한다. 융복합 모드와 코딩 모드 사이의 절환은 측정 결과와 같이 소스 코드가 1,000라인 내외일 경우 순공학 시 평균 184[ms], 역공학 시 평균 362[ms] 정도로 1초 이내에서 순식간에 이루어지기 때문에 대기 시간을 거의 느끼지 못할 정도이다.

이처럼 공정을 융복합하는 하이브리드 방법은 프로그램 설계 모델링의 효율성과 효과성을 강력하게 증대시켜 준다.

검증 결과를 통해, SETL을 사용하면 시간 효율성을 크게 향상시켜 줄 수 있으며 이를 통해 개발 생산성 및 유지보수성을 증대시켜줄 수 있음을 확인하였다.

6. 결론 및 향후 과제

본 논문에서는 컴포넌트 기반의 설계 모델링을 위한 SOC과 이를 지원하는 C언어와 Java언어 버전 중에서 자동화 도구인 C언어 버전의 SETL의 연구 개발 성과를 중심으로 결과를 제시하였다.

이 방법을 실제 프로젝트에 적용하면 개발 생산성과 유지 보수성의 획기적인 증대가 가능할 것으로 예상된다. 특히 융복합 모드에서 작업을 하면 코딩 모드에서 작업을 하는 것보다 빠르고 정확하게 소프트웨어 개발 및 유지보수를 수행할 수 있다.

한편, SETL의 여러 고급 프로그래밍 언어에의 적용을 위한 작업은 아직 진행 중에 있다. 따라서 다음과 같은 점을 중심으로 하는 심화 연구가 필요하다.

- SETL이 현존하는 모든 고급 프로그래밍 언어를 지원할 수 있도록 기능 확장을 위한 연구
- 비즈니스 모델 및 분석 모델까지 융복합의 범위를 확대하여 조직의 의사 결정 수준에서 실무적인 프로그래밍 작업 수준까지 통합하기위한 방법에 관한 연구
- SETL의 연구 개발 과정에서 축적한 기술을 활용하여 소프트웨어 품질의 정량적인 측정과 개선을 보다 체계적으로 수행하기 위한 연구 등

위와 같은 중점 사항들을 포함하여 지속적인 추가 연구를 바탕으로 향후 프로젝트 실무에의 적용을 통한 성공 사례를 축적해 나가고자 한다.

REFERENCES

[1] Hartree and Douglas, Calculating Instruments and Machines, The University of Illinois Press, pp. 112, 1949.

[2] Taub, Abraham, and John von Neumann, Collected Works 5., Macmillan. pp. 80-151, 1963.

[3] Corrado Böhm and Giuseppe Jacopini, Flow Diagrams, Turing Machines And Languages With Only Two Formation Rules, Communications of the ACM, Vol. 9, No. 5, pp. 366 ~ 371, May 1966.

[4] Edsger W. Dijkstra, Letters to the Editor (Go To Statement Considered Harmful), Communications of the ACM, Vol. 11, No. 3, pp. 147 ~ 148, March 1968.

[5] Donald E. Knuth, Structured Programming with go to Statements, Computing Surveys, Vol. 6, No. 4, pp. 261 ~ 301, December 1974.

[6] M. Hamilton and S. Zeldin, Top-Down, Bottom-Up, Structured Programming, and Program Structuring, Charles Stark Draper Laboratory, Document E-2728. Cambridge, Mass : Massachusetts Institute of Technology, December 1972.

[7] I. Nassi and B. Shneiderman, Flowchart techniques for structured programming, SIGPLAN Notices, ACM, Vol. 8, Issue 8, pp. 12 ~ 26, August 1973.

[8] Nakamura Sadatoshi, Ohara Shiqeyuki, and Odaka Akio, Classification of Control Macros of TS chart [in Japanese], Information Processing Society of Japan (IPSJ), pp. 1563-1564, Oct 1989.

[9] Robert W. Witty, Dimensional flowcharting, Journal of Software : Practice and Experience, Vol 7, Issue 5, pp. 553-584, September / October 1977.

[10] O. Ferstl, Flowcharting by Stepwise Refinement,

ACM SIGPLAN Notices, Vol. 13, No. 1, pp. 34-42, January 1978.

[11] ISO / IEC, ISO / IEC 8631, International Standard, Second edition, 1989-08-01, August 1989.

[12] Y. Futamura, and others, Development of Computer Programs by PAD (Problem Analysis Diagram), Proceedings of the Fifth International Software Engineering Conference. New York : IEEE ComputerSociety, pp. 325-332, 1981.

[13] Mikio Aoyama, Kazuyasu Miyamoto, Noritoshi Murakami, Hironobu Nagano, and Yoshihiro Oki, Design Specification in Japan : Tree-Structured Charts, Software, IEEE, Vol. 6, Issue 2, pp. 31-37, March 1989.

[14] C. Jinshong Hwang, Structured D-chart : A diagrammatic methodology in structured programming, AFIPS '82 : Proceedings of the June 7-10, 1982, national computer conference.

[15] Ken Orr, Structured Systems Development, New York : YOURDON Press, 1977.

[16] Jean-Dominique Warnier, Logical Construction of Programs, 3rd ed., translated by B. Flanagan. NewYork : Van Nostrand Reinhold 1976.

[17] Martin, J. and McClure, C., Diagramming Techniques for Analysts and Programmers, Prentice-Hall, Englewood Cliffs, NJ 1985.

[18] Grady Booch, Ivar Jacobson, and Jim Rumbaugh, (2000) OMG Unified Modeling Language Specification [dead link, Version 1.3 First Edition : March 2000. Retrieved 12 August 2008.

[19] OMG, Catalog of OMG Modeling and Metadata Specifications, http://www.omg.org/technology/documents/modeling_spec_catalog.htm. Retrieved 2012-02-21.

[20] OMG, OMG Unified Modeling Language (OMG UML) Superstructure, Version 2.4.1, August 2011.

[21] Raman Ramsin and Richard F. Paige, Process-Centered Review of Object Oriented Software Development Methodologies, ACM Computing Surveys, Vol. 40, No. 1, Article 3, Feb. 2008.

[22] Hong-Jun Yoo, A Study on the Evolved UML Behavioral Modeling Using CPD, Paper for Master's Degree, Dept. of Computer Eng. The Graduate School of Information & Communications Sungkyunkwan University, Oct. 2001.

[23] Yongchang Ren, Tao Xing, Xuguang Chai, Qiang Quan, and Xiaoji Chen, Study of Using Critical Path Method to Formulate the Algorithm of Software Project Schedule Planning, Information Management, Innovation Management and Industrial Engineering (ICIII) 2010 International Conference on, Vol. 4, pp. 126-129, Nov. 2010.

[24] Yoshio Sato and Yohsuke Hosoda, Development of a Web System for Making Problem Analysis Diagram (PAD), Systems Man and Cybernetics (SMC) 2010 IEEE International Conference on, pp. 1108-1114, Oct. 2010.

[25] Len Horton, Tools Are an Alternative to Playing Computer, Software Magazine, pp. 58-67, Jan. 1988.

[26] Carma McClure, THE THREE Rs OF SOFTWARE ATOMATION, PRENTICE-HALL 1992.

[27] Jean-Marc Desharnais and Alain April, Software maintenance productivity and maturity, Proceedings of the 11th International Conference on Product Focused Software, Jun 2010.

유 홍 준(Yoo, Hong Jun)

- 1999년 3월 : 일본 산노우대학 경영정보학과 졸업(학사)
- 2002년 2월 : 성균관대학교 정보통신대학원 컴퓨터공학과 졸업(석사)
- 2004년 2월 : 성균관대학교 일반대학원 전기전자 및 컴퓨터 공학과 박사과정 수료
- 2014년 4월 : 호서대학교 벤처대학원 정보경영학과 박사과정 재학중
- 2003년 11월 ~ 2014년 10월: ㈜한국정보감리평가원 재직
- 2014년 10월 ~ 현재 : ㈜소프트웨어품질기술원 재직
- 관심분야 : 가시적 소프트웨어공학(특히, 순공학, 역공학, 재구조화, 융복합 개발, 병렬 개발, 소프트웨어 품질보증과 평가 및 프로젝트관리, PMO, 감리)
- E-Mail : samtaeguk@naver.com

양 해 술(Yang, Hae Sool)

- 1975년 2월 : 홍익대학교 전기공학과 졸업(학사)
- 1978년 8월 : 성균관대학교 정보처리학과 졸업(석사)
- 1991년 3월 : 日本 오사카대학 정보공학과 SW공학 전공(공학박사)
- 2006년 2월 : Kazakhstan 유러시안 경제대학(명예경영학박사)
- 1975년 5월 ~ 1979년 6월 : 육군중앙경리단 전자계산실 시스템분석장교
- 1980년 3월 ~ 1995년 5월: 강원대학교 전자계산학과 교수
- 1986년 12월 ~ 1987년 12월: 日本 오사카대학 객원연구원
- 1995년 6월 ~ 2002년 12월: 한국소프트웨어품질연구소 소장
- 2010년 3월 ~ 2012년 2월: 호서대학교 창업대학원 원장
- 2012년 11월 : 대통령표창(SW산업발전유공) 수상
- 1999년 11월 ~ 현재 : 호서대학교 벤처대학원 교수
- 관심분야 : SW공학(특히, SW품질보증과 품질평가, 품질감리 및 컨설팅, SI), SW프로젝트관리, 품질경영)
- E-Mail : hsyang@hoseo.edu